Simple Schemas for Unordered XML

Iovka Boneva
University of Lille & INRIA, France
iovka.boneva@inria.fr

Radu Ciucanu
University of Lille & INRIA, France
radu.ciucanu@inria.fr

Sławek Staworko
University of Lille & INRIA, France
slawomir.staworko@inria.fr

ABSTRACT

We consider unordered XML, where the relative order among siblings is ignored, and propose two simple yet practical schema formalisms: *disjunctive multiplicity schemas* (DMS), and its restriction, *disjunction-free multiplicity schemas* (MS). We investigate their computational properties and characterize the complexity of the following static analysis problems: schema satisfiability, membership of a tree to the language of a schema, schema containment, twig query satisfiability, implication, and containment in the presence of schema. Our research indicates that the proposed formalisms retain much of the expressiveness of DTDs without an increase in computational complexity.

1. INTRODUCTION

When XML is used for *document-centric* applications, the relative order among the elements is typically important e.g., the relative order of paragraphs and chapters in a book. On the other hand, in case of *data-centric* XML applications, the order among the elements may be unimportant [1]. In this paper we focus on the latter use case. As an example, take a trivialized fragment of an XML document containing the DBLP repository in Figure 1. While the order of the elements *title*, *author*, and *year* may differ from one publication to another, it has no impact on the semantics of the data stored in this semi-structured database.

A *schema* for XML is a description of the type of admissible documents, typically defining for every node its *content model* i.e., the children nodes it must, may or cannot contain. For instance, in the DBLP example, we shall require every *article* to have exactly one *title*, one *year*, and one or more *author*'s. A *book* may additionally contain one *publisher* and may also have one or more *editor*'s instead of *author*'s. A schema has numerous important uses. For instance, it allows to validate a document against a schema and identify potential errors. A schema also serves as a reference for any user who does not know yet the structure of the XML document and attempts to query or modify its contents.

The *Document Type Definition (DTD)*, the most widespread XML schema formalism for (ordered) XML [6, 13], is essentially a set of rules associating with each label a regular expression that defines the admissible sequences of children. The DTDs are best fitted towards ordered content because they use regular expressions, a formalism that defines sequences of labels. However, when unordered content model needs to be defined, there is a tendency to use *over-permissive* regular expressions. For instance, the DTD below corresponds to the one used in practice for the DBLP repository:

$$\begin{aligned} dblp &\rightarrow (article \mid book)^* \\ article &\rightarrow (title \mid year \mid author)^* \\ book &\rightarrow (title \mid year \mid author \mid editor \mid publisher)^* \end{aligned}$$

This DTD allows an *article* to contain any number of *title*, *year*, and *author* elements. A *book* may also have any number of *title*, *year*, *author*, *editor*, and *publisher* elements. These regular expressions are clearly over-permissive because they allow documents that do not follow the intuitive guidelines set out earlier e.g., a document containing an *article* with two *title*'s and no *author* should not be admissible.

While it is possible to capture unordered content models with regular expressions, a simple pumping argument shows that their size may need to be exponential in the number of possible labels of the children. In case of the DBLP repository, this number reaches values up to 12, which basically precludes any practical use of such regular expressions. This suggests that over-permissive regular expressions may be employed for the reasons of conciseness and readability.

The use of over-permissive regular expressions, apart from allowing documents that do not follow the guidelines, has other negative consequences e.g., in static analysis tasks that involve the schema. Take for example the following two twig queries [2, 23]:

$$/dblp/book[author = \text{“}C.\ Papadimitriou\text{”}]$$
$$/dblp/book[author = \text{“}C.\ Papadimitriou\text{”}][title]$$

The first query selects the elements labeled *book*, children of *dblp* and having an *author* containing the text “*C. Papadimitriou.*” The second query additionally requires that *book* has a *title*. Naturally, these two queries should be

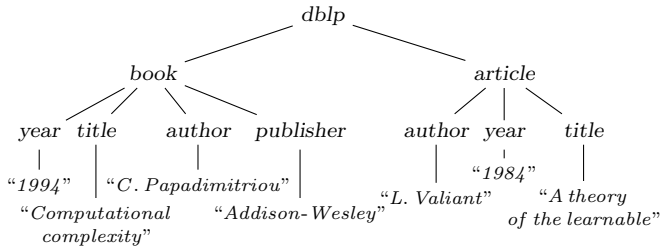


Figure 1: A trivialized DBLP repository.

Copyright is held by the author/owner. Sixteenth International Workshop on the Web and Databases (WebDB 2013), June 23, 2013 - New York, NY, USA.

Problem of interest	*DTD*	*DMS*	disjunction-free *DTD*	*MS*
Schema satisfiability	PTIME [9, 21]	PTIME (Prop. 4.3)	PTIME [9, 21]	PTIME (Prop. 4.3)
Membership	PTIME [9, 21]	PTIME (Prop. 4.3)	PTIME [9, 21]	PTIME (Prop. 4.3)
Schema containment	PSPACE-c†/PTIME [9, 21]	PTIME (Th. 4.2)	coNP-h†/PTIME [9, 16]	PTIME (Th. 4.2)
Query satisfiability‡	NP-c [4]	NP-c (Prop. 4.4)	PTIME [4]	PTIME (Th. 4.7)
Query implication‡	EXPTIME-c [19]	EXPTIME-c (Prop. 4.4)	PTIME (Cor. 4.9)	PTIME (Th. 4.7)
Query containment‡	EXPTIME-c [19]	EXPTIME-c (Prop. 4.4)	coNP-c (Cor. 4.9)	coNP-c (Th. 4.8)

† when non-deterministic regular expressions are used. ‡ for twig queries.

Table 1: Summary of complexity results.

equivalent because every *book* element should have a *title* child. However, the DTD above does not capture properly this requirement, and, consequently, the two queries are not equivalent w.r.t. this DTD.

In this paper, we study two new schema formalisms: the *disjunctive multiplicity schema (DMS)* and its restriction, the *disjunction-free multiplicity schema (MS)*. While they use a user-friendly syntax inspired by DTDs, they define unordered content model only, and, therefore, they are better suited for unordered XML. A DMS is a set of rules associating with each label the possible number of occurrences for all the allowed children labels by using *multiplicities*: "$*$" (0 or more occurrences), "+" (1 or more), "?" (0 or 1), "1" (exactly 1 occurrence; often omitted for brevity). Additionally, alternatives can be specified using restricted *disjunction* ("|") and all the conditions are gathered with *unordered concatenation* ("‖"). For instance, the following DMS captures precisely the intuitive requirements for the DBLP repository:

$$\begin{array}{rcl} \mathit{dblp} & \rightarrow & \mathit{article}^{*} \,\|\, \mathit{book}^{*} \\ \mathit{article} & \rightarrow & \mathit{title} \,\|\, \mathit{year} \,\|\, \mathit{author}^{+} \\ \mathit{book} & \rightarrow & \mathit{title} \,\|\, \mathit{year} \,\|\, \mathit{publisher}^{?} \,\|\, (\mathit{author}^{+} \mid \mathit{editor}^{+}) \end{array}$$

In particular, an *article* must have exactly one *title*, exactly one *year*, and at least one *author*. A *book* may additionally have a *publisher* and may have one or more *editor*'s instead of *author*'s. Note that, unlike the DTD defined earlier, this DMS does not allow documents having an *article* with several *title*'s or without any *author*.

There has been an attempt to use DTD-like rule based schemas to define unordered content models by interpreting the regular expressions under *commutative closure* [3]: essentially, an unordered collection of children matches a regular expression if there exists an ordering that matches the regular expression in the standard way. However, testing whether there exists a permutation of a word that matches a regular expression is NP-complete [15], which implies a significant increase in computational complexity of the membership problem i.e., validating an XML document against the schema. The schema formalisms proposed in this paper, DMS and MS, can be seen as DTDs interpreted under commutative closure using restricted classes of regular expressions. Two natural questions arise: do these restrictions allow us to avoid the increase in computational complexity, and how much of the expressiveness of DTDs is retained. The answers are generally positive. There is no increase in computational complexity but also no decrease (cf. Table 1). In particular, similarly to DTDs, the membership test can be performed in streaming manner (Section 4.1). Furthermore, the proposed schema formalisms seem to capture a significant part of the expressiveness of DTDs used in practice (Section 5).

We study the complexity of several basic decision problems: schema satisfiability, membership of a tree to the language of a schema, containment of two schemas, twig query satisfiability, implication, and containment in the presence of schema. Table 1 contains the summary of complexity results compared with general DTDs and disjunction-free DTDs. The lower bounds for the decision problems for DMS and MS are generally obtained with easy adaptations of their counterparts for general DTDs and disjunction-free DTDs. To obtain upper bounds we develop several new tools. *Dependency graphs* for MS and a generalized definition of an *embedding* of a query help us to reason about query satisfiability, query implication, and query containment in the presence of MS. An alternative characterization of DMS with *characterizing triples of sets* is used to reduce the containment of DMS to the containment of the sets of their triples, which can be tested in PTIME. We add that our constructions and results for MS extend easily to disjunction-free DTDs and allow to solve the problems of query implication and query containment, which, to the best of our knowledge, have not been previously studied for disjunction-free DTDs.

Because of space restriction, the proofs of all claims are omitted, they can be found in the appendix of the full version of the paper available at `http://arxiv.org/abs/1303.4277`.

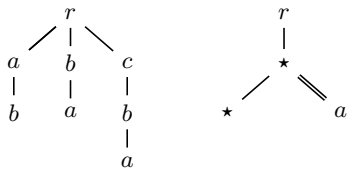
Related work. Languages of unordered trees can be expressed by *logic formalisms* or by *tree automata*. Boneva et al. [7, 8] make a survey on such formalisms and compare their expressiveness. The fundamental difference resides in the kind of constraints that can be expressed for the allowed collections of children for some node. We mention here only formalisms introduced in the context of XML. *Presburger automata* [22], *sheaves automata* [11], and the *TQL logic* [10] allow to express *Presburger constraints* on the numbers of occurrences of the different symbols among the children of some node. This is also equivalent to considering DTDs under commutative closure, similarly to [3]. The consequence of the high expressive power is that the membership problem is NP-complete for an unbounded alphabet [15]. Therefore, these formalisms were not extensively used in practice. Suitable restrictions on Presburger automata and on the TQL logic allow to obtain the same expressiveness as the MSO logic on unordered trees [7, 8]. DMS are strictly less expressive than these MSO-equivalent languages. Static analysis problems involving twig queries were not studied for these languages. Additionally, we believe that DMS are more appropriate to be used as schema languages, as they were designed as such, in particular regarding the more user-friendly DTD-like syntax. As mentioned earlier, unordered content model can also be defined by DTDs defining commutatively-closed sets of ordered trees. An (ordered) tree matches such a DTD iff

all tree obtained by reordering of sibling nodes also matches the DTD. This also turns out to be equally expressive as MSO on unordered trees [7, 8]. However, such a DTD may be of exponential size w.r.t. the size of the alphabet and, moreover, it is PSPACE-complete to test whether a DTD defines a commutatively-closed set of trees [18], which makes such DTDs unusable in practice. XML Schema allow for a bounded number of symbols to appear in arbitrary order, and RELAX NG allows to interleave sequences of symbols of bounded length. In contrast, the Kleene star in DMS allows for unbounded unordered collections of children. Schematron allows to specify very general constraints on the number of occurrences of symbols among the children of a node, in particular Presburger constraints are expressible. Schema languages using regular expressions with unbounded interleaving were studied in [12]. These are more expressive than DMS but exhibit high computational complexity of inclusion [12] and membership [5]. To the best of our knowledge, the static analysis problems involving queries were not studied for these languages when unordered content is allowed.

2. PRELIMINARIES

Throughout this paper we assume an alphabet Σ which is a finite set of symbols.

Trees. We model XML documents with unordered labeled trees. Formally, a *tree* t is a tuple $(N_t, root_t, lab_t, child_t)$, where N_t is a finite set of nodes, $root_t \in N_t$ is a distinguished root node, $lab_t : N_t \to \Sigma$ is a labeling function, and $child_t \subseteq N_t \times N_t$ is the parent-child relation. We assume that the relation $child_t$ is acyclic and require every non-root node to have exactly one predecessor in this relation. By *Tree* we denote the set of all finite trees.



(a) Tree t_0. (b) Twig query q_0.

Figure 2: A tree and a twig query.

Queries. We work with the class of twig queries, which are essentially unordered trees whose nodes may be additionally labeled with a distinguished wildcard symbol $\star \notin \Sigma$ and that use two types of edges, child (/) and descendant (//), corresponding to the standard XPath axes. Note that the semantics of //-edge is that of a proper descendant (and not that of descendant-or-self). Formally, a *twig query* q is a tuple $(N_q, root_q, lab_q, child_q, desc_q)$, where N_q is a finite set of nodes, $root_q \in N_q$ is the root node, $lab_q : N_q \to \Sigma \cup \{\star\}$ is a labeling function, $child_q \subseteq N_q \times N_q$ is a set of child edges, and $desc_q \subseteq N_q \times N_q$ is a set of descendant edges. We assume that $child_q \cap desc_q = \varnothing$ and that the relation $child_q \cup desc_q$ is acyclic and we require every non-root node to have exactly one predecessor in this relation. By *Twig* we denote the set of all twig queries. Twig queries are often presented using the abbreviated XPath syntax [23] e.g., the query q_0 in Figure 2(b) can be written as $r/\star[\star]//a$.

Embeddings. We define the semantics of twig queries using the notion of embedding which is essentially a mapping of nodes of a query to the nodes of a tree that respects the semantics of the edges of the query. Formally, for a query $q \in Twig$ and a tree $t \in Tree$, an *embedding* of q in t is a function $\lambda : N_q \to N_t$ such that:

1. $\lambda(root_q) = root_t$,
2. for every $(n, n') \in child_q$, $(\lambda(n), \lambda(n')) \in child_t$,
3. for every $(n, n') \in desc_q$, $(\lambda(n), \lambda(n')) \in (child_t)^+$ (the transitive closure of $child_t$),
4. for every $n \in N_q$, $lab_q(n) = \star$ or $lab_q(n) = lab_t(\lambda(n))$.

If there exists an embedding from q to t we say that t *satisfies* q and we write $t \models q$. By $L(q)$ we denote the set of all the trees satisfying q. Note that we do not require the embedding to be injective i.e., two nodes of the query may be mapped to the same node of the tree. Figure 3 presents all embeddings of the query q_0 in the tree t_0 from Figure 2.

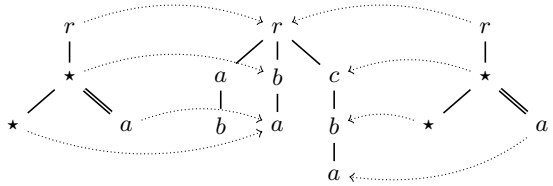


Figure 3: Embeddings of q_0 in t_0.

Unordered words. An *unordered word* is essentially a multiset of symbols i.e., a function $w : \Sigma \to \mathbb{N}_0$ mapping symbols from the alphabet to natural numbers, and we call the number $w(a)$ the number of occurrences of the symbol a in w. We also write $a \in w$ as a shorthand for $w(a) \neq 0$. An empty word ε is an unordered word that has 0 occurrences of every symbol i.e., $\varepsilon(a) = 0$ for every $a \in \Sigma$. We often use a simple representation of unordered words, writing each symbol in the alphabet the number of times it occurs in the unordered word. For example, when the alphabet is $\Sigma = \{a, b, c\}$, $w_0 = aaacc$ stands for the function $w_0(a) = 3$, $w_0(b) = 0$, and $w_0(c) = 2$.

The (unordered) concatenation of two unordered words w_1 and w_2 is defined as the multiset union $w_1 \uplus w_2$ i.e., the function defined as $(w_1 \uplus w_2)(a) = w_1(a) + w_2(a)$ for all $a \in \Sigma$. For instance, $aaacc \uplus abbc = aaaabbccc$. Note that ε is the identity element of the unordered concatenation $\varepsilon \uplus w = w \uplus \varepsilon = w$ for all unordered word w. Also, given an unordered word w, by w^i we denote the concatenation $w \uplus \ldots \uplus w$ (i times).

A *language* is a set of unordered words. The unordered concatenation of two languages L_1 and L_2 is a language $L_1 \uplus L_2 = \{w_1 \uplus w_2 \mid w_1 \in L_1, w_2 \in L_2\}$. For instance, if $L_1 = \{a, aac\}$ and $L_2 = \{ac, b, \varepsilon\}$, then $L_1 \uplus L_2 = \{a, ab, aac, aabc, aaacc\}$.

3. MULTIPLICITY SCHEMAS

A *multiplicity* is an element from the set $\{*, +, ?, 0, 1\}$. We define the function $[\![\cdot]\!]$ mapping multiplicities to sets of natural numbers. More precisely:

$$[\![*]\!] = \{i \in \mathbb{N} \mid i \geqslant 0\}, \quad [\![+]\!] = \{i \in \mathbb{N} \mid i \geqslant 1\}, \quad [\![?]\!] = \{0, 1\},$$
$$[\![1]\!] = \{1\}, \quad [\![0]\!] = \{0\}.$$

Given a symbol $a \in \Sigma$ and a multiplicity M, the language of a^M, denoted $L(a^M)$, is $\{a^i \mid i \in [\![M]\!]\}$. For example, $L(a^+) = \{a, aa, \ldots\}$, $L(b^0) = \{\varepsilon\}$, and $L(c^?) = \{\varepsilon, c\}$.

A *disjunctive multiplicity expression* E is:

$$E := D_1^{M_1} \parallel \ldots \parallel D_n^{M_n},$$

where for all $1 \leqslant i \leqslant n$, M_i is a multiplicity and each D_i is:

$$D_i := a_1^{M'_1} \mid \ldots \mid a_k^{M'_k},$$

where for all $1 \leqslant j \leqslant k$, M'_j is a multiplicity and $a_j \in \Sigma$. Moreover, we require that every symbol $a \in \Sigma$ is present at most once in a disjunctive multiplicity expression. For instance, $(a \mid b) \| (c \mid d)$ is a disjunctive multiplicity expression, but $(a \mid b) \parallel c \parallel (a \mid d)$ is not because a appears twice. A *disjunction-free multiplicity expression* is an expression which uses no disjunction symbol "|" i.e., an expression of the form $a_1^{M_1} \parallel \ldots \parallel a_k^{M_k}$, where for all $1 \leqslant i \leqslant k$, the a_i's are pairwise distinct symbols in the alphabet and the M_i's are multiplicities.

The language of a disjunctive multiplicity expression is:

$$L(a_1^{M_1} \mid \ldots \mid a_k^{M_k}) = L(a_1^{M_1}) \cup \ldots \cup L(a_k^{M_k}),$$
$$L(D^M) = \{w_1 \uplus \ldots \uplus w_i \mid w_1, \ldots, w_i \in L(D) \wedge i \in [\![M]\!]\},$$
$$L(D_1^{M_1} \parallel \ldots \parallel D_n^{M_n}) = L(D_1^{M_1}) \uplus \ldots \uplus L(D_n^{M_n}).$$

When a symbol a (resp. a disjunctive multiplicity expression E) has multiplicity 1, we often write a (resp. E) instead of a^1 (resp. E^1). Moreover, we omit writing symbols and disjunctive multiplicity expressions with multiplicity 0. Take for instance, $E_0 = a^+ \parallel (b \mid c) \parallel d^?$ and note that both the symbols b and c as well as the disjunction $(b \mid c)$ have an implicit multiplicity 1. The language of E_0 is:

$$L(E_0) = \{a^i b^j c^k d^\ell \mid i, j, k, \ell \in \mathbb{N},\ i \geqslant 1,\ j + k = 1,\ \ell \leqslant 1\}.$$

Next, we formally define the proposed schema formalisms.

Definition 3.1 *A* disjunctive multiplicity schema (DMS) *is a tuple $S = (root_S, R_S)$, where $root_S \in \Sigma$ is a designated root label and R_S maps symbols in Σ to disjunctive multiplicity expressions. By DMS we denote the set of all disjunctive multiplicity schemas. A* disjunction-free multiplicity schema (MS) *$S = (root_S, R_S)$ is a restriction of the DMS, where R_S maps symbols in Σ to disjunction-free multiplicity expressions. By MS we denote the set of all disjunction-free multiplicity schemas.*

To define satisfiability of a DMS (or MS) S by a tree t we first define the unordered word ch_t^n of children of a node $n \in N_t$ of t i.e., $ch_t^n(a) = |\{m \in N_t \mid (n, m) \in child_t \wedge lab_t(m) = a\}|$. Now, a tree t *satisfies* S, in symbols $t \models S$, if $lab_t(root_t) = root_S$ and for any node $n \in N_t$, $ch_t^n \in L(R_S(lab_t(n)))$. By $L(S) \subseteq Tree$ we denote the set of all the trees satisfying S.

In the sequel, we represent a schema $S = (root_S, R_S)$ as a set of rules of the form $a \to R_S(a)$, for any $a \in \Sigma$. If $L(R_S(a)) = \varepsilon$, then we write $a \to \epsilon$ or we simply omit writing such a rule.

Example 3.2 *We present schemas S_1, S_2, S_3, S_4 illustrating the formalisms defined above. They have the root label r and the rules:*

S_1 :	$r \to a \parallel b^* \parallel c^?$	$a \to b^?$	$b \to a^?$	$c \to b$
S_2 :	$r \to c \parallel b \parallel a$	$a \to b^?$	$b \to a$	$c \to b$
S_3 :	$r \to (a \mid b)^+ \parallel c$	$a \to b^?$	$b \to a^?$	$c \to b$
S_4 :	$r \to (a \mid b \mid c)^*$	$a \to \epsilon$	$b \to a^?$	$c \to b$

S_1 and S_2 are MS, while S_3 and S_4 are DMS. The tree t_0 from Figure 2(a) satisfies only S_1 and S_3.

4. STATIC ANALYSIS

We first define the problems of interest and we formally state the corresponding decision problems parameterized by the class of schema and, when appropriate, by a class of queries.

Schema satisfiability – checking if there exists a tree satisfying the given schema:

$$\mathrm{SAT}_{\mathcal{S}} = \{S \in \mathcal{S} \mid \exists t \in Tree.\ t \models S\}.$$

Membership – checking if the given tree satisfies the given schema:

$$\mathrm{MEMB}_{\mathcal{S}} = \{(S, t) \in \mathcal{S} \times Tree \mid t \models S\}.$$

Schema containment – checking if every tree satisfying one given schema satisfies another given schema:

$$\mathrm{CNT}_{\mathcal{S}} = \{(S_1, S_2) \in \mathcal{S} \times \mathcal{S} \mid L(S_1) \subseteq L(S_2)\}.$$

Query satisfiability by schema – checking if there exists a tree that satisfies the given schema and the given query:

$$\mathrm{SAT}_{\mathcal{S},\mathcal{Q}} = \{(S, q) \in \mathcal{S} \times \mathcal{Q} \mid \exists t \in L(S).\ t \models q\}.$$

Query implication by schema – checking if every tree satisfying the given schema satisfies also the given query:

$$\mathrm{IMPL}_{\mathcal{S},\mathcal{Q}} = \{(S, q) \in \mathcal{S} \times \mathcal{Q} \mid \forall t \in L(S).\ t \models q\}.$$

Query containment in the presence of schema – checking if every tree satisfying the given schema and one given query also satisfies another given query:

$$\mathrm{CNT}_{\mathcal{S},\mathcal{Q}} = \{(p, q, S) \in \mathcal{Q} \times \mathcal{Q} \times \mathcal{S} \mid \forall t \in L(S).\ t \models p \Rightarrow t \models q\}.$$

We next study these decision problems for DMS an MS.

4.1 Disjunctive multiplicity schema

In this section we present the complexity results for DMS. We first introduce an alternative representation of schemas used to establish the complexity of schema containment. Recall that $a \in w$ means that $w(a) \neq 0$. Given a disjunctive multiplicity expression E, we define the (*characterizing*) *triple* (C_E, N_E, P_E) of E consisting of the following sets:

- The *conflicting pairs of siblings* C_E consisting of pairs of symbols in Σ such that E defines no word using both symbols simultaneously:

$$C_E = \{(a_1, a_2) \in \Sigma \times \Sigma \mid \nexists w \in L(E).\ a_1 \in w \wedge a_2 \in w\}.$$

- The *extended cardinality map* N_E captures for each symbol in the alphabet the possible numbers of its occurrences in the unordered words defined by E:

$$N_E = \{(a, w(a)) \in \Sigma \times \mathbb{N} \mid w \in L(E)\}.$$

- The *sets of required symbols* P_E which captures symbols that must be present in every word; essentially, a set of symbols X belongs to P_E if every word defined by E contains at least one element from X:

$$P_E = \{X \subseteq \Sigma \mid \forall w \in L(E).\ \exists a \in X.\ a \in w\}.$$

As an example we take $E_0 = a^+ \parallel (b \mid c) \parallel d^?$. Because P_E is closed under supersets, we list only its minimal elements:

$$C_{E_0} = \{(b, c), (c, b)\}, \qquad P_{E_0} = \{\{a\}, \{b, c\}, \ldots\},$$
$$N_{E_0} = \{(b, 0), (b, 1), (c, 0), (c, 1), (d, 0), (d, 1), (a, 1), (a, 2), \ldots\}.$$

The triples allow us to capture the containment of disjunctive multiplicity expressions:

Lemma 4.1 *For all disjunctive multiplicity expressions E_1 and E_2, $L(E_2) \subseteq L(E_1)$ iff $C_{E_1} \subseteq C_{E_2}$, $N_{E_2} \subseteq N_{E_1}$, and $P_{E_1} \subseteq P_{E_2}$.*

The above lemma also shows that two equivalent DMS yield the same triples and hence the triple (C_E, N_E, P_E) can be viewed as the normal form of a given expression E. We point out that N_E may be infinite but it can be represented in a compact manner using multiplicities: for all letter a, the set $\{x \in \mathbb{N} \mid (a, x) \in N_E\}$ is representable by a multiplicity. Also, P_E may be exponential in the size of Σ but it can be represented with its $\subseteq$-minimal elements. Note that C_E has a number of elements quadratic in the number of letters, but allows for a representation linear in the size of Σ. These compact representations allow to easily test inclusion. Using Lemma 4.1, this allows us to establish one of our main results:

Theorem 4.2 CNT_{DMS} *is in PTIME.*

Next we present complexity results for satisfiability and membership. A streaming algorithm processes an XML document in a single pass, and it is earliest since it outputs its result at the earliest point. For a tree t, $height(t)$ is the height of t defined in the usual way. We employ the standard RAM model and assume that subsequent natural numbers are used as labels in Σ.

Proposition 4.3 *Checking satisfiability of a DMS S can be done in time $O(|\Sigma|^2)$. There exists an earliest streaming algorithm that checks membership of a tree t in a DMS S in time $O(|\Sigma| \times |t| + |\Sigma|^2)$ and using space $O(height(t) \times |\Sigma| + |\Sigma|^2)$.*

For satisfiability, the algorithm performs a preprocessing in time $O(|\Sigma|^2)$, and then performs a simple process based on dynamic programming to check that S is satisfiable. For checking the membership of a tree t to the language of S, the input tree t is given in XML format. The algorithm works for any arbitrary ordering of sibling nodes. In a preprocessing stage, the algorithm constructs (compact representations of) the triples of expressions used by the schema S and requires $O(|\Sigma|^2)$ space. [1] During the execution the algorithm maintains a stack whose height is the depth of the currently visited node. The algorithm stores on the stack a vector mapping $a \in \Sigma$ to a value among 0, 1, or 2^+ indicating that the current node has none, one or more occurrences of the symbol a among its children. Naturally, the bound on space required for stack operations is $O(height(t) \times |\Sigma|)$. The algorithm is earliest and rejects a tree as soon as the opening tag is read for nodes that violate either some conflicting pair or the allowed cardinality.

We end the section with three complexity results that follow from known facts.

Proposition 4.4 $\mathrm{SAT}_{DMS,Twig}$ *is NP-complete.* $\mathrm{IMPL}_{DMS,Twig}$ *and* $\mathrm{CNT}_{DMS,Twig}$ *are EXPTIME-complete.*

Proof sketch. Query satisfiability for DTDs is NP-complete [4] and can be adapted to DMS. Complexity of query implication and query containment follow from the EXPTIME-completeness of twig query containment in presence of DTDs [19]. □

[1]Remark that, as DMS forbids repetition of symbols, the size of the representation of any expression is linear in $|\Sigma|$.

4.2 Disjunction-free multiplicity schema

In this section we present the complexity results for MS. Although query satisfiability and query implication are intractable for DMS, these problems become tractable for MS because they can be reduced to testing embedding of queries in some dependency graphs that we define in the sequel. Recall that MS use expressions of the form $a_1^{M_1} \parallel \ldots \parallel a_n^{M_n}$.

Definition 4.5 *Given an MS $S = (root_S, R_S)$, the* dependency graph *of S is a directed rooted graph $G_S = (\Sigma, root_S, E_S)$ with the node set Σ, where $root_S$ is the distinguished root node and $(a, b) \in E_S$ if $R_S(a) = \ldots \parallel b^M \parallel \ldots$ and $M \in \{*, +, ?, 1\}$. Furthermore, the edge (a, b) is called* nullable *if $0 \in [\![M]\!]$ (i.e., M is $*$ or $?$), otherwise (a, b) is called* non-nullable *(i.e., M is $+$ or 1). The* universal dependency graph *of an MS S is the subgraph G_S^{u} containing only the non-nullable edges.*

In Figure 4 we present the dependency graphs for the schema S_5 containing the rules $r \to a^+ \parallel b^*,\ a \to b^?,\ b \to \epsilon$.

Figure 4: Dependency graph G_{S_5} and universal dependency graph $G_{S_5}^{\mathrm{u}}$ for schema S_5.

An MS S is *pruned* if G_S^{u} is acyclic. We observe that any MS has an equivalent pruned version which can be constructed in PTIME by removing the rules for the labels from which a cycle can be reached in the universal dependency graph. Note that a schema is satisfiable iff no cycle can be reached from its root in the universal dependency graph. From now on, we assume w.l.o.g. that all the MS that we manipulate are pruned.

We generalize the notion of embedding as a mapping of the nodes of a query q to the nodes of a rooted graph $G = (\Sigma, root, E)$, which can be either a dependency graph or a universal dependency graph. Formally, an *embedding* of q in G is a function $\lambda : N_q \to \Sigma$ such that:

1. $\lambda(root_q) = root$,
2. for every $(n, n') \in child_q$, $(\lambda(n), \lambda(n')) \in E$,
3. for every $(n, n') \in desc_q$, $(\lambda(n), \lambda(n')) \in E^+$ (the transitive closure of E),
4. for every $n \in N_q$, $lab_q(n) = \star$ or $lab_q(n) = \lambda(n)$.

If there exists an embedding from q to G, we write $G \preccurlyeq q$. The dependency graphs and embeddings capture satisfiability and implication of queries by MS.

Lemma 4.6 *For a twig query q and an MS S we have: 1) q is satisfiable by S iff $G_S \preccurlyeq q$, 2) q is implied by S iff $G_S^{\mathrm{u}} \preccurlyeq q$.*

Furthermore, testing the embedding of a query in a graph can be done in polynomial time with a simple bottom-up algorithm. Hence,

Theorem 4.7 $\mathrm{SAT}_{MS,Twig}$ *and* $\mathrm{IMPL}_{MS,Twig}$ *are in PTIME.*

The intractability of the containment of twig queries [17] implies the coNP-hardness of the containment of twig queries in the presence of MS. Proving the membership of the problem to coNP is, however, not trivial. Given an instance

(p, q, S), the set of all the trees satisfying p and S can be characterized with a set $\mathcal{G}(p, S)$ containing an exponential number of polynomially-sized graphs and p is contained in q in the presence of S iff the query q can be embedded into all the graphs in $\mathcal{G}(p, S)$. This condition is easily checked by a non-deterministic Turing machine.

Theorem 4.8 $\mathrm{CNT}_{MS,Twig}$ *is coNP-complete.*

We also point out that the results are easily adapted to disjunction-free DTDs, which allows us to state results which, to the best of our knowledge, are novel.

Corollary 4.9 $\mathrm{IMPL}_{disj\text{-}free\text{-}DTD,Twig}$ *is in PTIME and* $\mathrm{CNT}_{disj\text{-}free\text{-}DTD,Twig}$ *is coNP-complete.*

5. EXPRESSIVENESS OF DMS

We compare the expressive power of DMS and DTDs with focus on schemas used in real-life applications. First, we introduce a simple tool for comparing regular expressions with disjunctive multiplicity expressions, and by extension, DTDs with DMS. For a regular expression R, the language $L(R)$ of unordered words is obtained by removing the relative order of symbols from every ordered word defined by R. A disjunctive multiplicity expression E *captures* R if $L(E) = L(R)$. A DMS S *captures* a DTD D if for every symbol the disjunctive multiplicity expression on the rhs of a rule in S captures the regular expression on the rhs of the corresponding rule in D. We believe that this simple comparison is adequate because if a DTD is to be used in a data-centric application, then supposedly the order between siblings is not important. Therefore, a DMS that captures a given DTD defines basically the same type of admissible documents, without imposing any order among siblings. Naturally, if we use the above notion to compare the expressive powers of DTDs and DMS, DTDs are strictly more expressive than DMS.

We use the comparison on the XMark [20] benchmark and the University of Amsterdam XML Web Collection [13]. We find that all 77 regular expressions of the XMark benchmark are captured by DMS rules, and among them 76 by MS rules. As for the DTDs found in the University of Amsterdam XML Web Collection, 84% of regular expressions (with repetitions discarded) are captured by DMS rules and among them 74.6% by MS rules. Moreover, 55.5% of full DTDs in the collection are captured by DMS and among them 45.8% by MS. Note that these figures should be interpreted with caution, as we do not know which of the considered DTDs were indeed intended for data-centric applications. We believe, however, that these numbers give a generally positive answer to the question of how much of the expressive power of DTDs the proposed schema formalisms, DMS and MS, retain.

6. CONCLUSIONS AND FUTURE WORK

We have studied the computational properties and the expressive power of new schema formalisms, designed for unordered XML: the disjunctive multiplicity schema (DMS) and its restriction, the disjunction-free multiplicity schema (MS). DMS and MS can be seen as DTDs using restricted classes of regular expressions and interpreted under commutative closure to define unordered content models. These restrictions allow on the one hand to maintain a relatively low computational complexity of basic static analysis problems while retaining a significant part of expressive power of DTDs.

An interesting question remains open: are these the most general restrictions that allow to maintain a low complexity profile? We believe that the answer to this question is negative and intend to identify new practical features that could be added to DMS and MS. One such feature are *numeric occurrences* [14] of the form $a^{[n,m]}$ that generalize multiplicities by requiring the presence of at least n and no more than m elements a. It would also be interesting to see to what extent our results can be used to propose hybrid schemas that allow to define ordered content for some elements and unordered model for others.

7. REFERENCES

[1] S. Abiteboul, P. Bourhis, and V. Vianu. Highly expressive query languages for unordered data trees. In *ICDT*, pages 46–60, 2012.

[2] S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Tree pattern query minimization. *VLDB J.*, 11(4):315–331, 2002.

[3] C. Beeri and T. Milo. Schemas for integration and translation of structured and semi-structured data. In *ICDT*, pages 296–313, 1999.

[4] M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. *J. ACM*, 55(2), 2008.

[5] M. Berglund, H. Björklund, and J. Högberg. Recognizing shuffled languages. In *LATA*, pages 142–154, 2011.

[6] G. Bex, F. Neven, and J. Van den Bussche. DTDs versus XML Schema: A practical study. In *WebDB*, pages 79–84, 2004.

[7] I. Boneva and J. Talbot. Automata and logics for unranked and unordered trees. In *RTA*, pages 500–515, 2005.

[8] I. Boneva, J. Talbot, and S. Tison. Expressiveness of a spatial logic for trees. In *LICS*, pages 280–289, 2005.

[9] A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Inf. Comput.*, 142(2):182–206, 1998.

[10] L. Cardelli and G. Ghelli. TQL: a query language for semistructured data based on the ambient logic. *Mathematical Structures in Computer Science*, 14(3):285–327, 2004.

[11] S. Dal-Zilio and D. Lugiez. XML schema, tree logic and sheaves automata. In *RTA*, pages 246–263, 2003.

[12] W. Gelade, W. Martens, and F. Neven. Optimizing schema languages for XML: Numerical constraints and interleaving. *SIAM J. Comput.*, 38(5):2021–2043, 2009.

[13] S. Grijzenhout and M. Marx. The quality of the XML web. In *CIKM*, pages 1719–1724, 2011.

[14] P. Kilpeläinen and R. Tuhkanen. One-unambiguity of regular expressions with numeric occurrence indicators. *Inf. Comput.*, 205(6):890–916, 2007.

[15] E. Kopczynski and A. To. Parikh images of grammars: Complexity and applications. In *LICS*, pages 80–89, 2010.

[16] W. Martens, F. Neven, and T. Schwentick. Complexity of decision problems for XML schemas and chain regular expressions. *SIAM J. Comput.*, 39(4):1486–1530, 2009.

[17] G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1):2–45, 2004.

[18] F. Neven and T. Schwentick. XML schemas without order. 1999.

[19] F. Neven and T. Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *Logical Methods in Computer Science*, 2(3), 2006.

[20] A. Schmidt, F. Waas, M. Kersten, M. Carey, I. Manolescu, and R. Busse. XMark: A benchmark for XML data management. In *VLDB*, pages 974–985, 2002.

[21] T. Schwentick. Trees, automata and XML. In *PODS*, page 222, 2004.

[22] H. Seidl, T. Schwentick, and A. Muscholl. Numerical document queries. In *PODS*, pages 155–166, 2003.

[23] W3C. XML Path language (XPath) 1.0, 1999.